ILP-based Scheduling for Asynchronous Circuits in Bundled-Data Implementation

Hiroshi Saito
The University of Aizu
Japan
hiroshis@u-aizu.ac.jp

Nattha Jindapetch
Prince of Songkla University
Thailand
nattha.s@psu.ac.th

Tomohiro Yoneda
National Institute of Informatics
Japan
yoneda@nii.ac.jp

Chris Myers
The University of Utah
USA
myers@vlsigroup.ece.utah.edu

Takashi Nanya
The University of Tokyo
Japan
nanya@hal.rcast.u-tokyo.ac.jp

Abstract

In this paper, we propose a new scheduling method for asynchronous circuits in bundled-data implementation. The method is based on integer linear programming (ILP) which explores an optimum schedule under resource or time constraints. To schedule descriptions with many operations, our method approximate start times of operations and formulate an ILP based on the approximated start times. Because less numbers of variables and constraints are required compared to the traditional ILP formulation, the schedule of operations is determined in short time preserving the quality of resulting circuit.

1 Introduction

Different from synchronous circuits where circuit components are controlled by a global clock signal, circuit components in asynchronous circuits are controlled by local handshake signals. The use of local handshake signals leads to several advantages such as average-case performance improvement, low power consumption, and so on. However, the design of asynchronous circuits is difficult because hazard-freeness in implementation must be guaranteed. Nevertheless, only a limited number of CAD tools is available. In particular, support for behavioral synthesis which synthesizes a circuit from a behavioral description of an application is very limited [1, 2].

In this paper, we propose a new scheduling method for asynchronous circuits in bundled-data implementation. In bundled-data implementation, data operations are assumed to be executed in the maximum execution delays of used resources. Although several advantages in asynchronous circuits such as average-case performance are lost, the area required for bundled-data implementation is less than other implementations such as dual-rail implementation [3]. Moreover, the design of bundled-data implementation is easier than other implementations.

Scheduling is a process in behavioral synthesis to determine the start time of operations under resource or time constraints. The proposed scheduling method is based on *integer linear programming* (ILP) where an objective function such as the execution time or the number of required resources is minimized under constraints. Until now, many ILP-based scheduling algorithms have been developed for scheduling of operations in synchronous circuits. In those algorithms, ILPs are formulated by considering clock cycles. However, as there is no clock signal in asynchronous circuits, the use of the traditional ILP formulation for asynchronous circuits may waste scheduling time. This is a critical problem for ILP-based scheduling because it takes explosive computation time to solve an ILP. To deal with a behavioral description with many operations, we propose a new ILP formulation for asynchronous circuits in bundled-data implementation based on approximated start times of operations.

As related work, Badia et al. proposed a scheduling method based on heuristic list scheduling [1]. The method determines the schedule of operations under resource constraints by exploring the availability of resources and the completion of operations. Although it can deal with descriptions with many operations, the optimality is inferior compared to ILP-based scheduling. Bachman et al. proposed a scheduling method where resource sharing between operations is explored by introducing additional dependency between them [2]. However, it cannot deal with descriptions with many operations because of the huge computational complexity. We proposed a heuristic scheduling method for bundled-data implementation in [4]. Different from the heuristic approach, the method proposed in this paper can determine more optimum schedule.

The rest of this paper is organized as follows. Section 2 describes basic notions used in this paper. Section 3 describes a traditional ILP formulation. Section 4 proposes a new ILP formulation. Section 5 describes experimental results. Finally, Section 6 gives conclusions.

2 Basic Notions

2.1 Data Flow Graph

In this work, *data flow graphs* (DFGs) are used as an input of our scheduling algorithm.

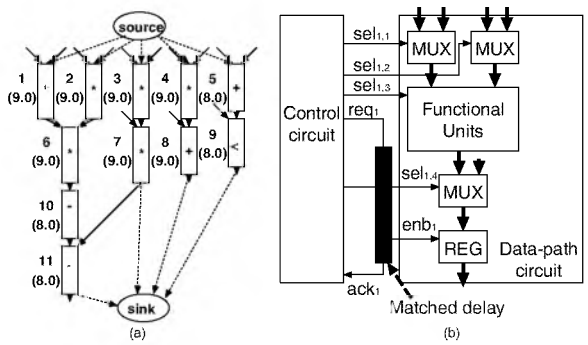

Figure 1. (a)DFG and (b)bundled-data implementation.

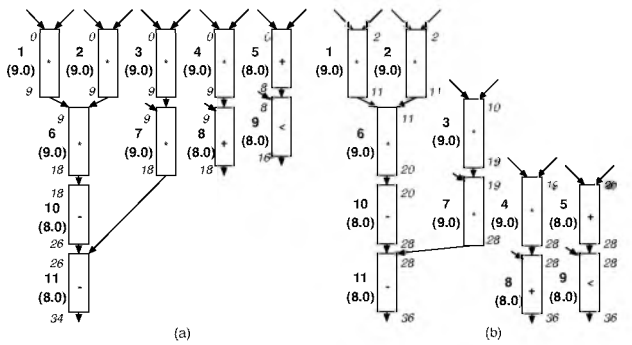

Figure 2. (a)ASAP schedule and (b)ALAP schedule.

A DFG defined by $G = \langle N, E \rangle$ represents the flow of data in an application. A node $i (i = 1, \ldots, n)$ in the node set N represents a data operation. Source node and sink node are also included in N as reference nodes. Those are ignored in many cases through the paper. An edge (i, j) in the edge set E represents a data dependency from node i to node j. Each node is labeled by the maximum execution delay d_i required to execute the corresponding operation. d_i is decided by binding data-path resources in a given resource library to node i. Fig.1(a) shows an example of DFGs.

2.2 Bundled-Data Implementation

The model of asynchronous circuits in bundled-data implementation is illustrated in Fig.1(b).

In bundled-data implementation, one data bit is represented by one wire. Therefore, resources used in the data-path circuit are the same as the ones used in synchronous circuits. The difference is only the control scheme. Local handshake signals are used instead of a global clock.

A set of data-path resources to execute corresponding operation for node i is controlled by the control circuit with a pair of handshake signals req_i and ack_i and selection signals $sel_{i,p} (p = 1, \ldots, q)$. Signal req_i initiates the operation of node i in the data-path circuit while signal ack_i indicates the completion of the operation to the control circuit. The completion of an operation is guaranteed by a delay element d_i called "matched delay" which is put on the wire of signal req_i. To guarantee the correctness of writing a data into a register, enable signal enb_i is generated from the matched delay.

3 Scheduling by ILP

Integer linear programming (ILP) is used to solve the scheduling problem in behavioral synthesis [5]. This section describes a traditional ILP formulation used for synchronous circuit design.

3.1 ASAP scheduling and ALAP scheduling

As soon as possible (ASAP) scheduling and *as late as possible* (ALAP) scheduling are applied to a given DFG to obtain a feasible schedule under a time constraint.

In ASAP scheduling, operations are scheduled so that they are executed as soon as possible. The critical path delay in a give DFG is identified. On the other hand, in ALAP scheduling, operations are scheduled under a time constraint so that they are executed as late as possible. The mobility of data operations under the time constraint is identified. Fig.2(a) and (b) show the ASAP schedule and the ALAP schedule for the DFG shown in Fig.1(a). The time constraint is assumed to be 36ns which is used for the calculation of the ALAP schedule. For convenience, we denote the ASAP start time of node i as $asaps_i$, the ASAP completion time as $asapc_i$, the ALAP start time as $alaps_i$, and the ALAP completion time as $alapc_i$.

3.2 Decision of Control Steps and Assignment Variables

Scheduling is the process to determine start times of operations. Start times are represented as "control steps" (steps). We denote the set of steps as L, a step as $l (l = 0, \ldots, m)$, and the start time of a step l as t_l.

In synchronous circuit design, steps imply clock cycles. Therefore, steps are derived from a given time constraint so that they have the same time interval. For example in Fig.3(a), the time constraint is assumed to be 36ns and the time interval between steps is assumed to be 2ns.

The set of steps where node i can be scheduled is denoted as S_i ($S_i = \{l \in L | asaps_i \leq t_l \leq alaps_i\}$). The first step and the last step in S_i are denoted as F_{S_i} and L_{S_i}. For example, S_9 for node 9 in Fig.3(a) is $\{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$. F_{S_i} is 4 and L_{S_i} is 14, respectively.

A binary assignment variable denoted as $x_{i,l}$ is assigned to each step $l \in S_i$. The variable $x_{i,l}$ is 1 when node i is scheduled at step l while it is 0 when node i is not scheduled. For example, assignment variables for node 9 are $x_{9,4}, \ldots, x_{9,14}$, respectively.

3.3 Objective Functions and Constraints

The objective function for resource constraint scheduling is represented as follows:

$$minimize : \sum_{l=F_{S_{sink}}}^{L_{S_{sink}}} t_l * x_{sink,l}. \quad (1)$$

This objective function means to minimize the start time of the sink node under resource constraints.

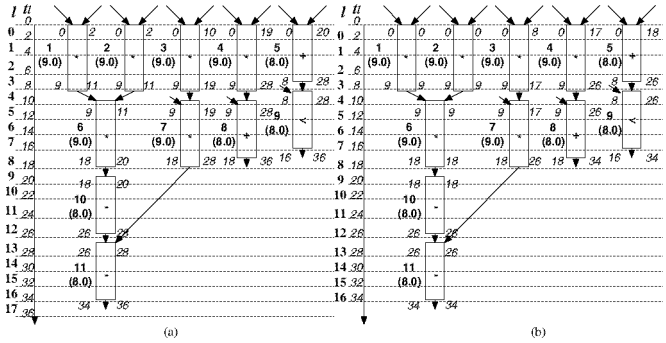

Figure 3. Steps: (a) time constraint is 36ns and (b) time constraint is 34ns.

Minimize
obj: 80 num_{comp} + 80 num_{add} + 80 num_{sub} + 480 num_{mult}
Subject To
// Node assignment constraints
c0: $x_{0,0} = 1$
c1: $x_{1,0} = 1$
c2: $x_{2,0} + x_{2,1} + x_{2,2} + x_{2,3} + 1$
:
// Dependency constraints
c11: $9\,x_{6,3} + 12\,x_{6,4} + 15\,x_{6,5} + 18\,x_{6,6} - (0\,x_{2,0} + 3\,x_{2,1} + 6\,x_{2,2} + 9\,x_{2,3})$ -9 >= 0
:
// Resource assginment constraints
c14: $x_{8,3}$ - num_{comp} <= 0;
c15: $x_{8,3} + x_{8,4} + x_{8,5}$ -num_{comp} <= 0;
:
c21: $x_{4,0}$ - num_{add} <= 0;
:

Figure 4. ILP formulation.

On the other hand, the objective function for time constraint scheduling is represented as follows:

$$minimize : \sum_{r \in R} c_r * num_r \tag{2}$$

where r represents a resource in a given resource library, R represents the set of resources, c_r represents the cost of resource r, and num_r represents the number of resource r. This objective function minimizes the total cost of data-path resources under time constraints.

These objective functions are solved under the following three types of constraints.

Node assignment constraints: Each node i must be scheduled in exactly one step $l \in S_i$. This constraint is represented as follows:

$$\sum_{l=F_{S_i}}^{L_{S_i}} x_{i,l} = 1, \forall i \in N \tag{3}$$

Dependency constraints: For each dependency (i, j) in a given DFG, the following dependency constraint must be satisfied.

$$\sum_{l=F_{S_j}}^{L_{S_j}} t_l * x_{j,l} - \sum_{l=F_{S_i}}^{L_{S_i}} t_l * x_{i,l} - d_i \geq 0, \forall (i,j) \in E \tag{4}$$

Resource assignment constraints: The number of operations executed by resource r at step l must be num_r or less. This constraint is represented as follows:

$$\sum_{i \in N_r} x_{i,l} - num_r \leq 0, \forall l \in L, \forall r \in R \tag{5}$$

where $i \in N_r$ means operations which are bound to resource r. Note that num_r is a constant value in resource constraint scheduling.

Fig.4 shows a part of the ILP formulation for the DFG shown in Fig.1(a) when the time constraint is assumed to be 36ns and the time interval is assumed to be 2ns. In this example, the number of used steps is 18 and the numbers of generated variables and constraints are 74 and 59.

4 ILP-Based Scheduling for Bundled-Data Implementations

4.1 Problem Definition

Although ILP-based scheduling algorithms can decide an optimum schedule for a given DFG under time or resource constraints, it takes substantial computation time to solve an ILP. Therefore, when a large number of constraints and assignment variables are derived from a given DFG, the algorithm may fail scheduling.

According to the ILP formulation, we understand that the number of constraints and assignment variables depends on the number of steps. Therefore, we may relax the time interval of steps to reduce the number of constraints and assignment variables. However, there is a situation that the time interval cannot be relaxed to keep a given time constraint. For example, suppose we assign the time interval between steps to be 2ns and the time constraint to be 34ns for the DFG shown in Fig.1. In this case, node 5 cannot be scheduled any step because no schedulable step exists between $asaps_5$ and $alaps_5$ (see Fig.3(b)). To enumerate schedulable steps for all nodes under this time constraint, we must assign the time interval to 1ns. Otherwise, we must change the time constraint.

The source of the problem comes from the fact that steps are decided so that they have the same time interval. Although the use of these steps is indispensable for scheduling of operations in synchronous circuits, it is inefficient for scheduling of operations in asynchronous circuits because asynchronous circuits do not use a global clock signal. This situation motivates us to develop alternative ILP formulation strategy.

Alternative ILP formulation comes from the nature of asynchronous circuits. In asynchronous circuits, operations are initiated by the completion of preceding operations. Therefore, the main idea of our proposed method is to decide steps based on approximated start times of operations. The start times of operations are easily approximated in bundled-data implementations because the delay of operations is fixed. After the decision of steps, approximated start times are directly converted to assignment variables. Further difference compared to the traditional ILP formulation is the generation of resource assignment constraints. Resource assignment constraints for a resource r are generated for a subset of steps which are distinguished by types of resources.

4.2 ILP Formulation based on Approximated Start Times

The procedure for ILP formulation consists of three steps. First, start times are approximated for each node. Then, steps and assignment variables used for ILP formulation are decided from approximated start times. Finally, an ILP is formulated by using decided steps and assignment variables.

4.2.1 Approximation of Start Times

Start times of each operation are approximated from completion times of preceding operations. Preceding operations are assumed to:

- Direct predecessors
- Operations which have no data dependence and can share the same resource

Although preceding operations are restricted, they are more practical restrictions. In many cases, operations are initiated by the completion of one of its predecessors. In addition, when resource sharing arises, operations are waited until the preceding operation that uses the same resource completes the execution.

Before the approximation of start times, we define several terminologies.

- ST_i - the set of start times for node i (a start time is represented by st).
- P_i - the set of direct predecessor nodes for node i
- C_i - the set of nodes which have no data dependence and can share the same resource as node i

Let us illustrate the approximation of start times using Fig.1(a). We approximate start times of node 8 from $asapc_i$ of preceding nodes. In this case, P_8 is $\{4\}$ and C_8 is $\{5\}$. Therefore, the start times of node 8 correspond to $asapc_4$ and $asapc_5$ (see Fig.5(a)). However, in this case, $asapc_5$ is ignored because $asapc_5$ is not in the range between $asaps_8$ and $alaps_8$. As a result, ST_8 is approximated to $\{9\}$.

The approximation considering only $asapc_j$ of a preceding node j in P_i and C_i is very restrictive. Therefore, the number of approximated start times for each node i is at most $|N|$. To approximate start times more, we consider the possible execution orders of more than two preceding nodes.

For example, we approximate start times of node 8 in Fig.1(a) from the possible execution order of two preceding nodes. P_8 and C_8 are $\{4\}$ and $\{5\}$, respectively. Then, we approximate start times for nodes 4 and 5. P_4, C_4, P_5, and C_5 are $\{source\}$, $\{1, 2, 3, 6, 7\}$, $\{source\}$, and $\{\emptyset\}$. Note that node 8 is not included in C_5 because start times of node 8 is the target. From $asapc_j$ of node j in P_4, C_4, and P_5, ST_4 is $\{0, 9, 18\}$ and ST_5 is $\{0\}$. Next, start times of node 8 are approximated from possible completion times of node 4 and node 5. Those are calculated by $st \in ST_4$ plus $delay_4$ and $st \in ST_5$ plus $delay_5$. As a result, ST_8 is approximated to $\{8, 9, 18, 27\}$. Because 8 and 27 are out of range between $asaps_8$ and $alaps_8$, ST_8 is finally $\{9, 18\}$. Fig.5(b) illustrates this approximation.

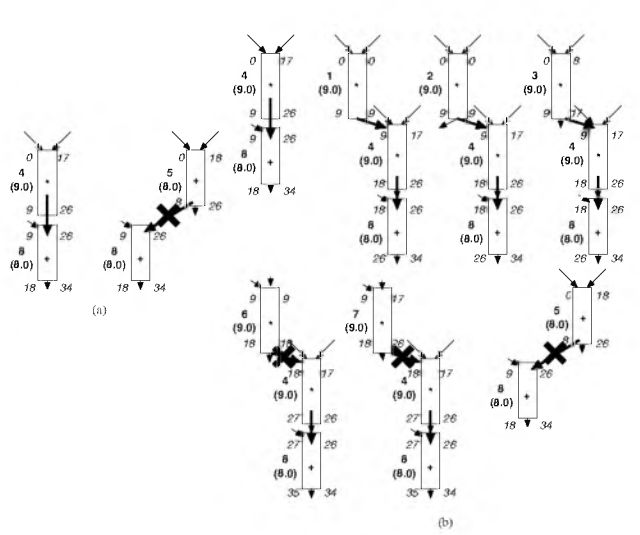

Figure 5. Approximation of start times: (a)from one preceding node and (b) from the possible execution order of two preceding nodes.

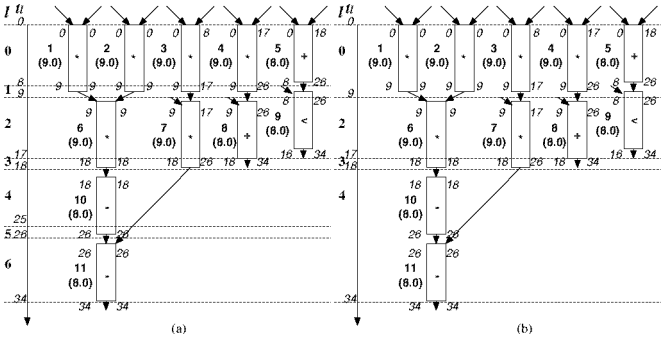

Figure 6. (a)Decided steps and (b)Steps for resource add.

Theoretically, all possible start times for node i can be approximated when we enumerate the possible execution order of $|N| - 1$ nodes. However, it takes so long time for approximation when $|N|$ is very large. Therefore, we implement the approximation method which enumerates the possible execution orders up to four nodes.

4.2.2 Decision of Steps and Assignment Variables from Approximated Start Times

After the approximation of start times, the union set of ST_i is taken. Then, start times in the union set are translated into steps. Fig.6(a) shows the decided steps for the DFG shown in Fig.1(a). Note that steps decided by the approximation of start times have different time intervals than steps decided by clock cycles. After steps are decided, each start time st in ST_i is translated into an assignment variable $x_{i,l}$.

4.2.3 ILP formulation

From decided steps and assignment variables, the objective function and constraints for ILP-based scheduling are formulated as explained in Section 3 except for resource

Table 1. Parameters of used resources.

Name	Area (# of Slices)	Delay [ns]
add	8	8
sub	8	8
comp	8	8
or	9	6
sfl	39	9
sfr	39	9
mul	48	9
div	96	18

assignment constraints. To generate resource assignment constraints, steps are distinguished by the type of resources. The set of steps, L_r, for a resource r consists of steps where a node bound to resource r can be scheduled. Consequently, constraint (5) is rewritten as follows:

$$\sum_{i \in N_r} x_{i,l} - num_r \leq 0, \forall l \in L_r, \forall r \in R \qquad (6)$$

Fig.6(b) shows steps for resource add. Because there is no operation bound to resource add that can be started from steps 1, 5, and 6, L_{add} is $\{0, 2, 3, 4\}$. Therefore, resource assignment constraints for resource add are generated for steps in L_{add}.

Compared to the traditional ILP formulation shown in Fig.4, 7 steps, 27 variables, and 23 constraints are generated. As the effect of our proposed ILP formulation, the numbers of steps, assignment variables, and constraints are reduced.

5 Experimental Results

In experiments, we show the effectiveness of our proposed ILP formulation for asynchronous circuits in bundled-data implementation. To show the effectiveness, we compare the number of steps, variables, and constraints between traditional ILP formulation and our proposed ILP formulation for several benchmarks. Then, we observe the effect of ILP formulations from the comparison of the scheduling time, latency, and the number of required resources. For comparison, we have implemented both ILP formulations in Java. With the selection of formulation strategies, designers can select either time constraint scheduling or resource constraint scheduling. As an ILP solver, CPLEX [6] has been used. Scheduling is carried out on an Windows machine with a Pentium4 processor (3.4GHz) and a 2G memory.

Table 1 shows resources used in the experiments with the parameters of delay and area. Those values are obtained by synthesizing Verilog HDL descriptions using Xilinx ISE ServicePack [7] and Vertex-II pro [8]. Note that the divider div cannot be synthesized. Therefore, we assumed the area and the delay to be two times more than the multiplier mul.

The experimental results of time constraint scheduling are shown in Table 2. For each benchmark, two rows are used to show the results of the traditional ILP formulation (every first row) and our proposed ILP formulation (every second row). The columns labeled with N and TC represent the number of nodes and the time constraint. The value of the column Int in every first row represents the time interval between steps. On the other hand, the value of the column Int in every second row represents the number of preceding nodes used for the enumeration of possible execution orders. When TC is set to the critical path of operations, the value of Int in every first row might be 1 (i.e., the time interval between steps was 1). Otherwise, there is a node whose start time is not approximated. The value of Int in every second row is decided so that the same scheduling result as the traditional one (i.e., the number of required resources) is obtained. The columns Step and FT represent the number of steps and the time of ILP formulation, respectively. The columns Var and Const represent the number of variables and constraints used in ILPs. The column RT represents the run time of CPLEX. We assume 1 hour as the limitation of CPLEX run time. All other columns with labels of resources represent the number of required resources. The symbol "-" means that the result can not be obtained because the run time of CPLEX exceeds 1 hour. The symbol "*" in the last column Opt represents that the optimum schedule is obtained. It means that the number of required resources is all one or equal to the one when the time interval in the traditional ILP formulation is set to be 1ns.

In time constraint scheduling, the run time of CPLEX is reduced in most of cases when our ILP formulation is used. This is because the number of constraints and variables in our ILP formulation is less than that of the traditional ILP formulation. In particular, the run time of CPLEX for our ILP formulation is very short in idctrow(3) even though the result could not be obtained in the traditional ILP formulation.

The experimental results of resource constraint scheduling are shown in Table 3. As before, two rows are used to show the results of the traditional ILP formulation (every first row) and our proposed ILP formulation (every second row). The column labeled with Ref represents a reference time to calculate ALAP schedule. The columns labeled with resource names represent constraints for resource numbers. As same as time constraint scheduling, the value of Int in every second row is decided so that the same scheduling result as the traditional one (i.e., the execution time) is obtained. The columns Step and FT represent the number of steps and the time of ILP formulation, respectively. The columns Var and Const represent the number of variables and constraints used in ILPs. The column RT represents the run time of CPLEX. As before, we assume 1 hour as the limitation of CPLEX run time. The column ET represents the execution time of operations obtained by scheduling. The symbol "*" in the last column Opt represents that the optimum schedule is obtained. It means that the execution time of operations is equal to the one when the time interval in the traditional ILP formulation is set to be 1ns.

In resource constraint scheduling, the run time of CPLEX is reduced in most of cases as time constraint scheduling when our ILP formulation is used. In particular, scheduling results for ewf(2) and sshu(2) are obtained within the time limitation where the results are not obtained in the traditional ILP formulation. However, the optimum result cannot be obtained in sshu(1) and idctrow(2) where the number of variables is less that that of the traditional ILP formulation. The reason directly comes from the fact that the approximation of start times is restricted. Another note is that the run time of CPLEX is also worse in these cases even though the number of variables is reduced. It is considered that the run time of CPLEX depends on the generated formula.

Table 2. Result of time constraint scheduling.

Name	N	TC [ns]	Int	Step	FT [s]	Var	Const	RT [s]	*add*	*sub*	*comp*	*or*	*sfl*	*sfr*	*mul*	*div*	Opt
diffeq	11	34	1	34	0.29	109	80	0.02	1	1	1				3		*
(1)			1	7	0.07	27	23	< 0.00	1	1	1				3		*
diffeq	11	51	3	17	0.05	108	67	0.08	1	1	1				2		*
(2)			1	13	0.08	40	35	0.05	1	1	1				2		*
diffeq	11	68	3	23	0.08	174	91	0.02	1	1	1				1		*
(3)			4	21	0.15	68	50	0.02	1	1	1				1		*
ewf	34	115	1	115	0.18	242	137	0.02	3						3		*
(1)			1	18	0.07	72	63	< 0.00	3						3		*
ewf	34	173	3	58	0.16	749	177	1.08	2						1		*
(2)			1	46	0.18	396	140	0.05	2						1		*
ewf	34	230	2	115	0.25	2082	284	1609.81	1						1		*
(3)			4	96	2.25	1241	228	1.38	1						1		*
sshu	42	206	1	206	0.38	964	419	0.06	2						2	2	*
(1)			1	52	0.16	174	133	< 0.00	2						2	2	*
sshu	42	309	3	103	0.26	1776	363	2.84	1						1	1	*
(2)			2	152	0.36	715	373	1.69	1						1	1	*
idctrow	67	84	1	84	0.24	1236	421	0.16	4	3		1	1	4	6		*
(1)			1	30	0.16	260	172	0.05	4	3		1	1	4	6		*
idctrow	67	126	3	42	0.18	1366	341	501.72	2	2		1	1	2	2		
(2)			2	60	0.39	879	301	6.06	2	2		1	1	2	2		
idctrow	67	168	3	56	0.21	2304	425	> 1h	-	-		-	-	-	-		
(3)			2	64	0.40	940	309	4.33	2	2		1	1	2	2		

Table 3. Result of resource constraint scheduling.

Name	N	Ref [ns]	*add*	*sub*	*comp*	*or*	*sfl*	*sfr*	*mul*	*div*	Int	Step	FT [s]	Var	Const	RT [s]	ET [ns]	Opt
diffeq	11	51	1	1	1				2		3	17	0.05	102	68	< 0.00	35	*
(1)			1	1	1				2		1	13	0.04	32	28	< 0.00	35	*
diffeq	11	68	1	1	1				1		3	23	0.07	173	92	0.03	62	*
(2)			1	1	1				1		4	22	0.13	67	37	0.02	62	*
ewf	34	173	2						1		3	58	0.16	756	182	12.48	132	*
(1)			2						1		1	46	0.19	403	142	1.28	132	*
ewf	34	230	1						1		2	115	0.24	2128	290	> 1h	-	
(2)			1						1		4	96	2.29	1290	232	0.64	223	
sshu	42	309	2						1	1	3	103	0.25	1799	365	0.56	210	*
(1)			2						1	1	4	200	1.32	1529	536	499.41	213	
sshu	42	309	1						1	1	3	103	0.40	1799	365	> 1h	-	
(2)			1						1	1	4	200	1.35	1529	537	1476.69	225	
idctrow	67	126	3	3		1	1	3	3		2	63	0.24	2034	441	232.77	102	
(1)			3	3		1	1	3	3		2	60	0.38	886	304	20.38	102	
idctrow	67	126	2	2		1	1	2	2		3	42	0.18	1369	350	172.67	113	
(2)			2	2		1	1	2	2		2	60	0.38	886	307	1394.72	119	

From these experiments, we can say that our ILP formulation is effective for asynchronous circuits in bundled-data implementation. Compared to the traditional ILP formulation, the schedule of operations can be determined in short time preserving the quality of resulting circuits. Because the scheduling problem is formulated with less number of constraints and variables, we expect that our proposed ILP formulation is particularly useful for large DFGs.

6 Conclusions

In this paper, we have proposed a new ILP-based scheduling method for asynchronous circuits in bundled-data implementation. In the method, the ILP formulation is carried out based on approximated start times of operations. Because less number of variables and constraints is required in most of cases compared to the traditional ILP formulation, the schedule of benchmarks including realistic examples has been determined in short time preserving the quality of resulting circuits.

As future work, we are going to extend our ILP formulation to deal with pipelining and other resources such as registers. We are also going to extend a traditional heuristic scheduling algorithm to deal with larger descriptions so that approximated start times are used for scheduling.

Acknowledgement

This work is supported by Grant-in-Aid for Scientific Research from Japan Society for the Promotion of Science (#18700047).

References

[1] R.Badia and J.Cortadella, High-Level Synthesis of Asynchronous Systems: Scheduling and Process Synchronization, *Proc. European Conference on Design Automation*, pp.70–74, 1993.

[2] B.Bachman, H.Zheng, and C.Myers, Architectural Synthesis of Timed Asynchronous Systems, *Proc. International Conference on Computer Designs*, pp.354–363, 1999.

[3] A.Davis and S.Nowick, An Introduction to Asynchronous Circuit Design, *Technical Report*, The University of Utah, UUCS-97-013, 1997.

[4] H.Saito, N.Jindapetch, T.Yoneda, C.Myers, and T.Nanya, A Scheduling Method for Asynchronous Bundled-Data Implementations, *Proc. International Workshop on Logic and Synthesis*, pp.341–348, 2005.

[5] C-T.Hwang, J-H.Lee, and Y-C.Hsu, A Formal Approach to the Scheduling Problem in High Level Synthesis, *IEEE Transactions on Computer-Aided Design*, vol.10, no.4, pp.464–475, 1991.

[6] ILOG Inc, CPLEX 9.1, http://www.ilog.com

[7] XILINX Inc, ISE ServicePack 8.1i http://www.xilinx.com

[8] XILINX Inc, Vertex-2 pro http://www.xilinx.com